Servlets Java

IUT Lyon 1

Yosra ZGUIRA

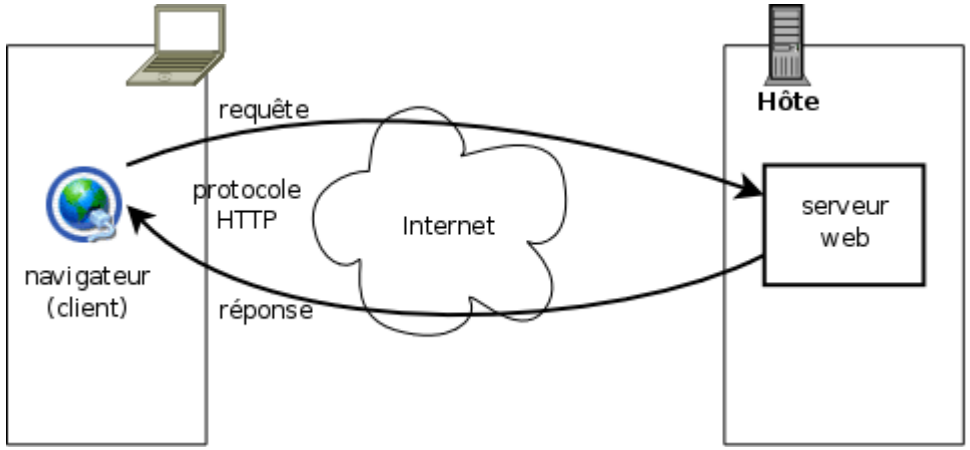
yosra.zguira@insa-lyon.fr

2016 - 2017

Introduction

- Le WEB fonctionne selon le modèle **client/serveur**
  - Via le **protocole HTTP**

- Cycle requête/réponse du protocole HTTP
  - Un client émet une requête vers un serveur
    - La requête possède le format HTTP
  - Le serveur est obligé de répondre au client
    - La réponse est formatée HTTP
  - Les requêtes sont indépendantes les unes des autres

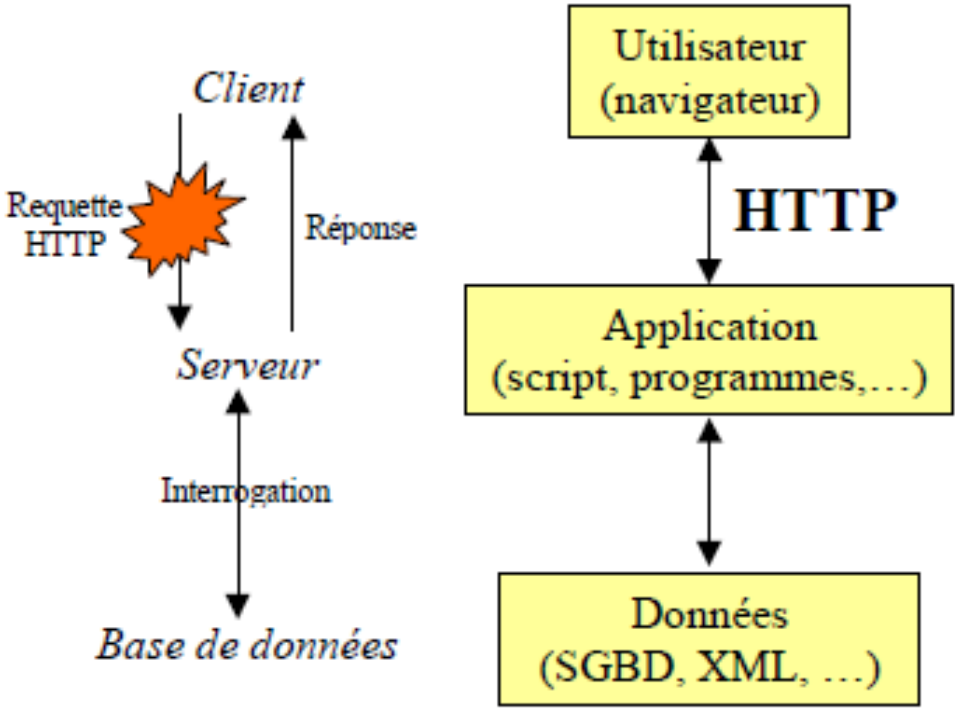


Contenu WEB dynamique

- **Pages WEB dynamiques pourquoi ?**
  - Adapter les pages à la demande du client (outils de recherche, …)
  - Présenter des informations changeantes (météo, actualités, catalogue de produit avec leur prix, ...)
- **Pages WEB dynamiques comment ?**
  - Des ressources statiques - texte, images, vidéo…,
  - Des ressources semi-statiques – SGBDR, BD XML
  - Des scripts et des programmes - CGI, PHP, JSP, Servlets

Pages dynamiques - architecture

- Au moins trois niveaux applicatifs:



Qu’est ce qu’une Servlet?

- **Servlet: Server-side applet**
- Une servlet est un composant Web **du coté serveur Web**:
  - qui permet d’étendre les possibilités d’un serveur Web
    - Possibilité de générer du contenu dynamique en réponse à des requêtes clients.
      - ➢ Un peu comme les scripts CGI (Common Gateway Interface).
  - C’est une classe Java exécutée sur un serveur multi-threadé
    - Elle est compilée sous forme de byte-code,
    - Elle est exécutée par une machine virtuelle Java (JVM).
  - Elle est mise en œuvre est gérée par un conteneur Web (Tomcat par exemple).

Caractéristiques d'une Servlet

- **Efficace**
  - Une servlet est un thread java (léger) et non un processus (lourd) de l'OS. C'est une instance unique, qui reste chargée en mémoire.
- **Pratique**
  - Existence d'une bibliothèque d'utilitaires très riche
- **Puissante**
  - Partage de données, communique avec son environnement
- **Portable**
  - S'exécute sur tous les OS et tous les serveurs
- **Sûre**
  - Pas de débordements de mémoire
- **Economique**
  - Il existe de nombreux serveurs gratuits ou peu chers

Servlet vs. Applet (1/2)

- **Applets :**
  - ✓ Interface graphique utilisateur,
  - ✓ Nécessite un browser adéquat,
  - ✓ Traitements sur le client (Client lourd),
  - ✓ Limites de services dues aux problèmes de sécurité.

- **Servlets:**
  - ✓ pas d'interface graphique utilisateur (Langage HTML),
  - ✓ pas de limitations de sécurité :
    - possibilité d'établissement de connexions avec d 'autres machines que le serveur (utilisation comme pont JDBC-ODBC),
    - possibilité d'appels systèmes (JDBC)
    - manipulation de ressources locales du serveur

Servlet vs. Applet (2/2)

- Avantage **Servlet**:
  - inhérents à Java :
    - JDK8 gratuit et portable
  - par rapport aux **Applets**:
    - plus facile à développer,
    - meilleures performances,
    - client léger.

Technologies serveur les plus connues

- **CGI** : Common Gateway Interface
  - programme de réponse écrit en n'importe quel langage
- **ASP** : Active Server Page
  - html, JavaScript, VBScript
  - Technologie propriétaire : *Microsoft*
- **PHP** :
  - Orienté SGBD, Technologie open source
- **Servlet**, **pages JSP** :
  - code basé sur Java, open source

Servlet vs. CGI (1/2)

- **Principe CGI** :
  - ✓ Un processus par requête est lancé sur le serveur.
- **Avantages CGI** :
  - ✓ Gratuit, pris en charge par tous les serveurs Web actuels,
  - ✓ Peut être écrit dans n'importe quel langage (C, perl).
- **Inconvénients CGI** :
  - ✓ Manque d'évolutivité (plusieurs processus créés),
  - ✓ Serveur très sollicité si plusieurs requêtes au même moment,
  - ✓ Assez lent et parfois difficile à développer.

Servlet vs. CGI (2/2)

- **Les Servlets sont portable, plus efficaces, plus pratiques et plus puissantes :**
  - ✓ indépendance des OS, c'est du Java !,
  - ✓ indépendance des serveurs web (Apache, Microsoft IIS, WebStar, etc.),
  - ✓ efficacité (connexion multi-threads avec les utilisateurs, un seul chargement, permanence en mémoire),
  - ✓ super API (Application Programming Interface) pour gérer les formulaires HTML,
  - ✓ dialogue possible avec des applets situées sur le client (utilisation d 'un protocole à objets distribués RMI),
  - ✓ gestion des sessions,
  - ✓ faible coût : kit de développement des Servlets gratuit (Apache/Tomcat reste la solution la plus efficace... 100% gratuite).

L'API Servlet (1/5)

- **L'API Servlet fournit un certain nombre de classes et d'interfaces permettant :**
  - ✓ le développement des Servlets,
  - ✓ leur déploiement,
  - ✓ et leur mise en œuvre au sein du conteneur Web.

- **L'API Servlet est contenue principalement dans deux packages :**
  - ✓ javax.servlet
  - ✓ javax.servlet.http

MapremièreServlet.java

```
import javax.servlet.*
import javax.servlet.http.*
public class ……{
…….
……
……
}
```

L'API Servlet (2/5)

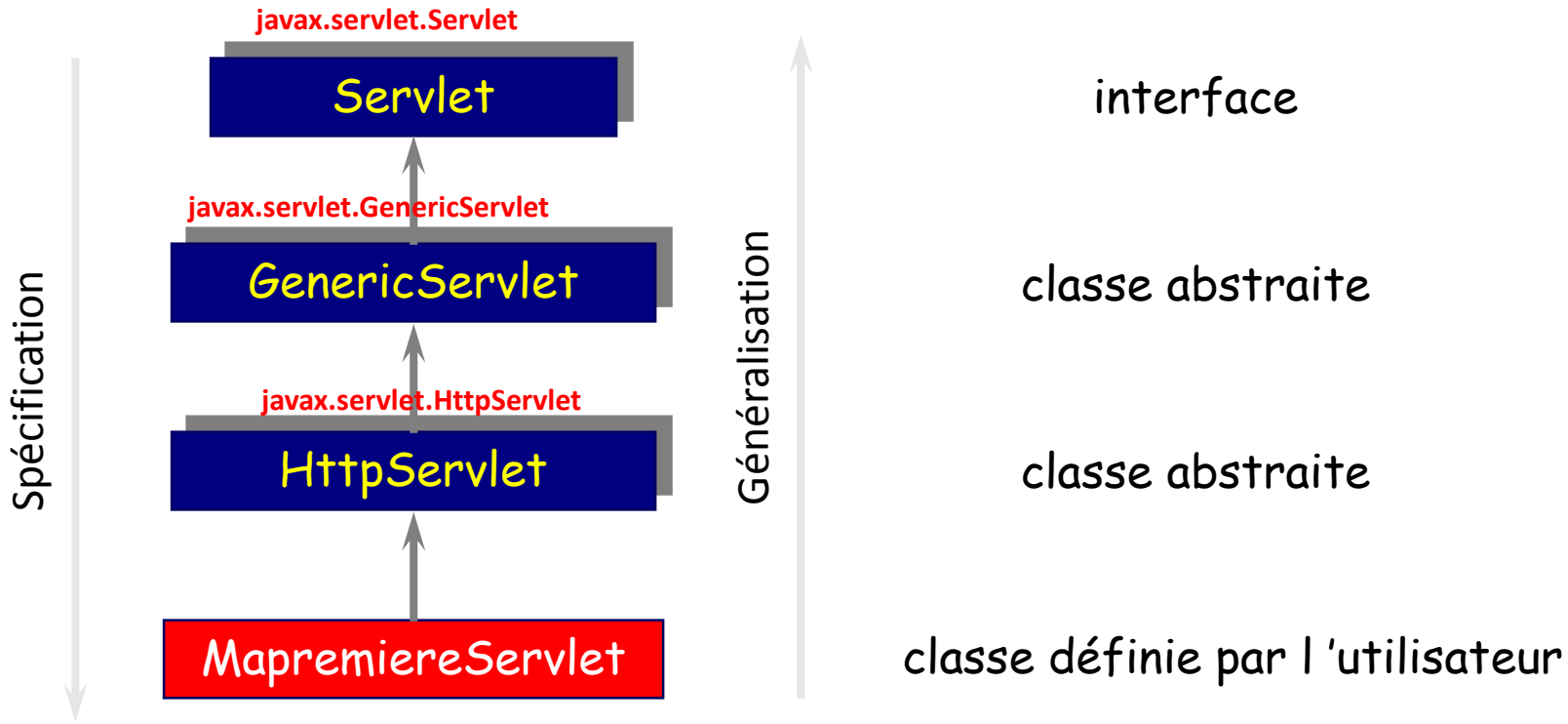
- Le package **javax.servlet** :
  - ✓ Contient les classes pour un support des Servlets génériques et indépendant du protocole.

- Le package **javax.servlet.http** :
  - ✓ Contient des extensions des classes du package javax.servlet.
    - Ces extensions consistent à ajouter des fonctionnalités spécifiques au protocole HTTP.

- **Le nom du package le plus haut « javax » au lieu du « java » plus familier, indique que l'API Servlet est une extension standard.**

L'API Servlet (3/5)

- Chaque Servlet utilisateur doit implémenter l'interface javax.servlet.Servlet soit directement soit par l'extension de la classe spéciale javax.servlet.GenericServlet ou javax.servlet.HttpServlet.



L'API Servlet (4/5)

- **L'interface javax.servlet.Servlet possède les méthodes :**
  - **init ( ) :**
    - ✓ pour initialiser la Servlet.
  - **Service ( )** :
    - ✓ pour recevoir et répondre aux requêtes des clients.
  - **destroy ()** :
    - ✓ détruire la servlet et ses ressources.
- **Ces 3 méthodes sont tous héritées donc par une Servlet utilisateur.**

L'API Servlet (5/5)

- **Une Servlet utilisateur peut implémenter l'interface javax.servlet.Servlet directement**

```
public class MapremiereServlet implements Servlet {
....
```

- **Une Servlet utilisateur indépendante du protocole devait être une sous classe de GenericServlet**

```
public class MapremiereServlet extends GenericServlet {
....
```

- **Une Servlet Http devait être une sous classe de HttpServlet.**

```
public class MapremiereServlet extends HttpServlet {
....
```

Modèle de programmation d'une Servlet (1/2)

- Une Servlet suit un modèle de programmation **requête-service-réponse.**

  ✓ A la place d'une méthode main( ), elle possède une méthode service( ) qui sera invoquer automatiquement à chaque fois que la Servlet reçoit une requête.

  ✓ La méthode service(objet1, objet2) accepte deux paramètres et permet de recevoir et de répondre aux requêtes des clients :

  - Un objet1: **javax.servlet.ServletRequest**

    - contient les informations nécessaires pour une communication du client vers le serveur (Requête).

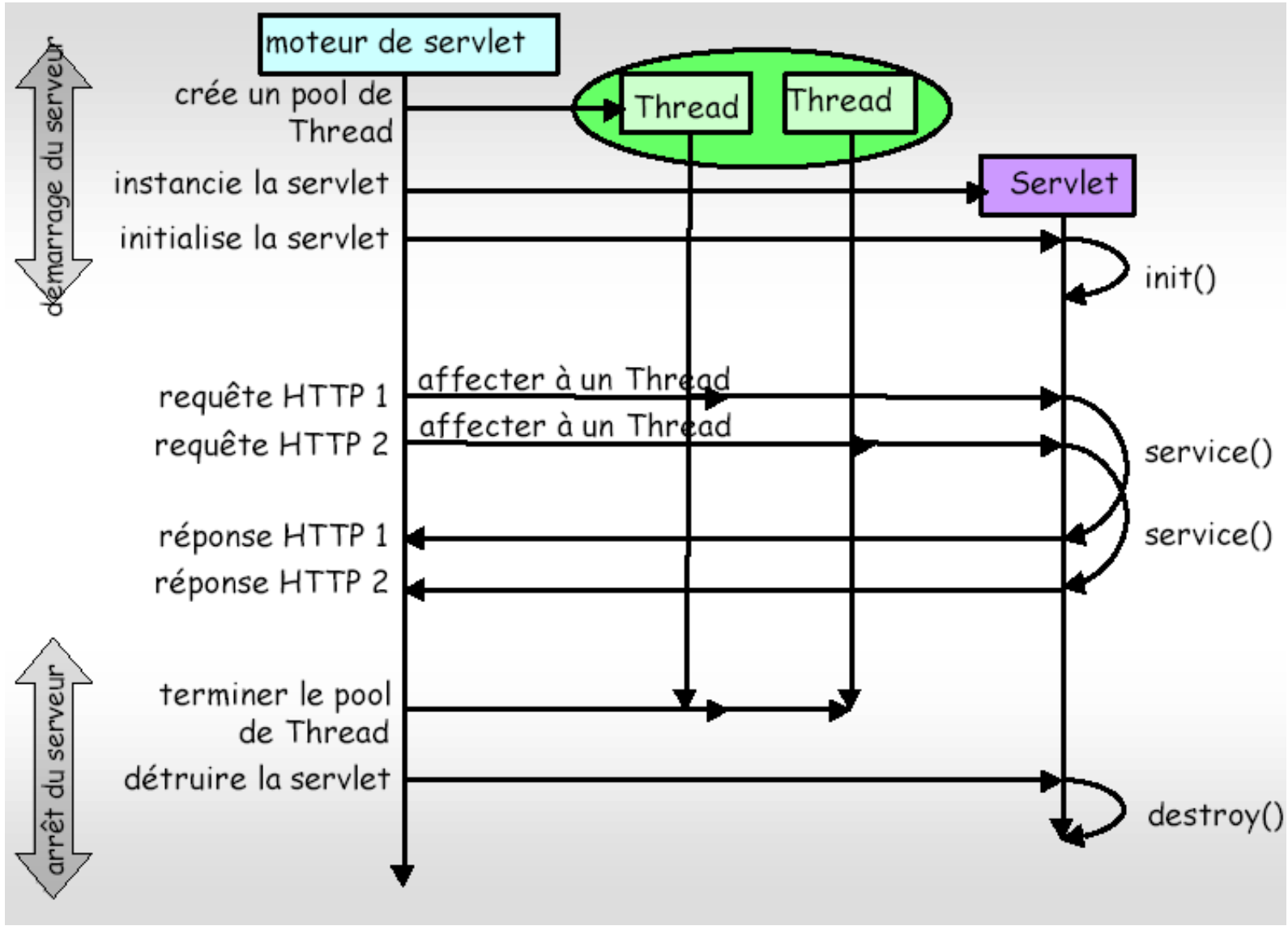
  - Un objet2: **javax.servlet.ServletResponse**

    -contient les informations nécessaires pour une communication du serveur vers le client (Réponse).

Modèle de programmation d'une Servlet (2/2)

- **Une Servlet générique doit surcharger sa méthode service(ServletRequest obj1, ServletResponse obj2).**

Cycle de vie d'une Servlet (1/2)



Cycle de vie d'une Servlet (2/2)

- Une *servlet* possède le cycle de vie suivant :

1. la *servlet* est crée puis initialisée (**init()**)
   - cette méthode n'est appelée par le serveur qu'une seule fois lors du chargement

2. traitements des requêtes (**service()**)
   - cette méthode est appelée automatiquement par le serveur à chaque requête de client

3. la *servlet* est détruite (**destroy()**)
   - cette méthode n'est appelée par le serveur qu'une seule fois à la fin, elle permet de libérer des ressources

Une servlet = plusieurs Threads

- Plusieurs Threads peuvent traverser la méthode **service()** en même temps

  ✓ Les variables doivent être locales aux méthodes

  ✓ Les attributs de classes ne peuvent être que de constantes de configuration initialisées dans la méthode init()

Structure de base d'une Servlet

```
import javax.servlet.*;

public class First implements Servlet {

   public void init(ServletConf config) throws ServletException {…}


   public void service(ServletRequest req, ServletResponse rep)
          throws ServletException, IOException {…}


   public void destroy() {…}
}
```

Modèle de programmation d'une Servlet HTTP (1/2)

- Une Servlet **Http** ne surcharge pas la méthode **service().**

✓ Cette méthode **service()** de la classe mère est remplacée avantageusement par 2 méthodes ayant la même signature :

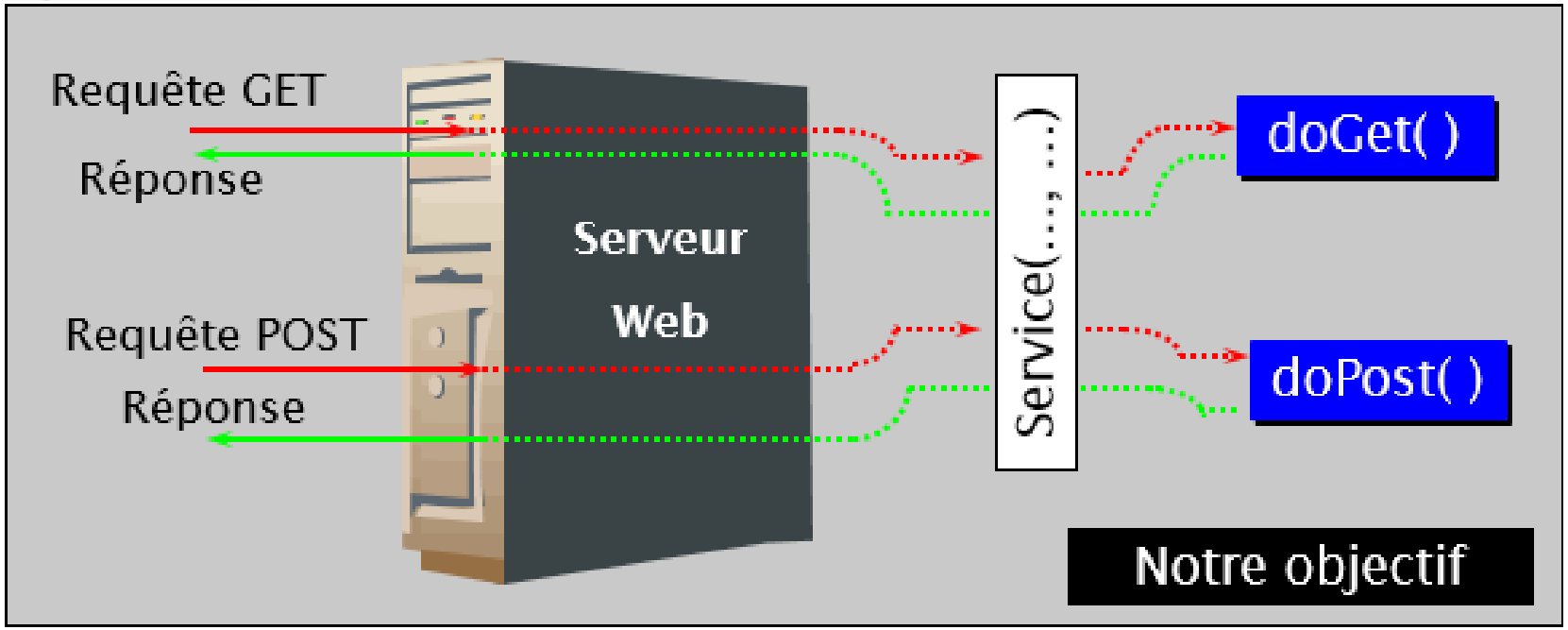
- **doGet()** : pour traiter des requêtes Http de type GET
- **doPost()** : pour traiter des requêtes Http de type POST

✓ Une Servlet Http doit obligatoirement contenir l'une ou l'autre de ces 2 méthodes.

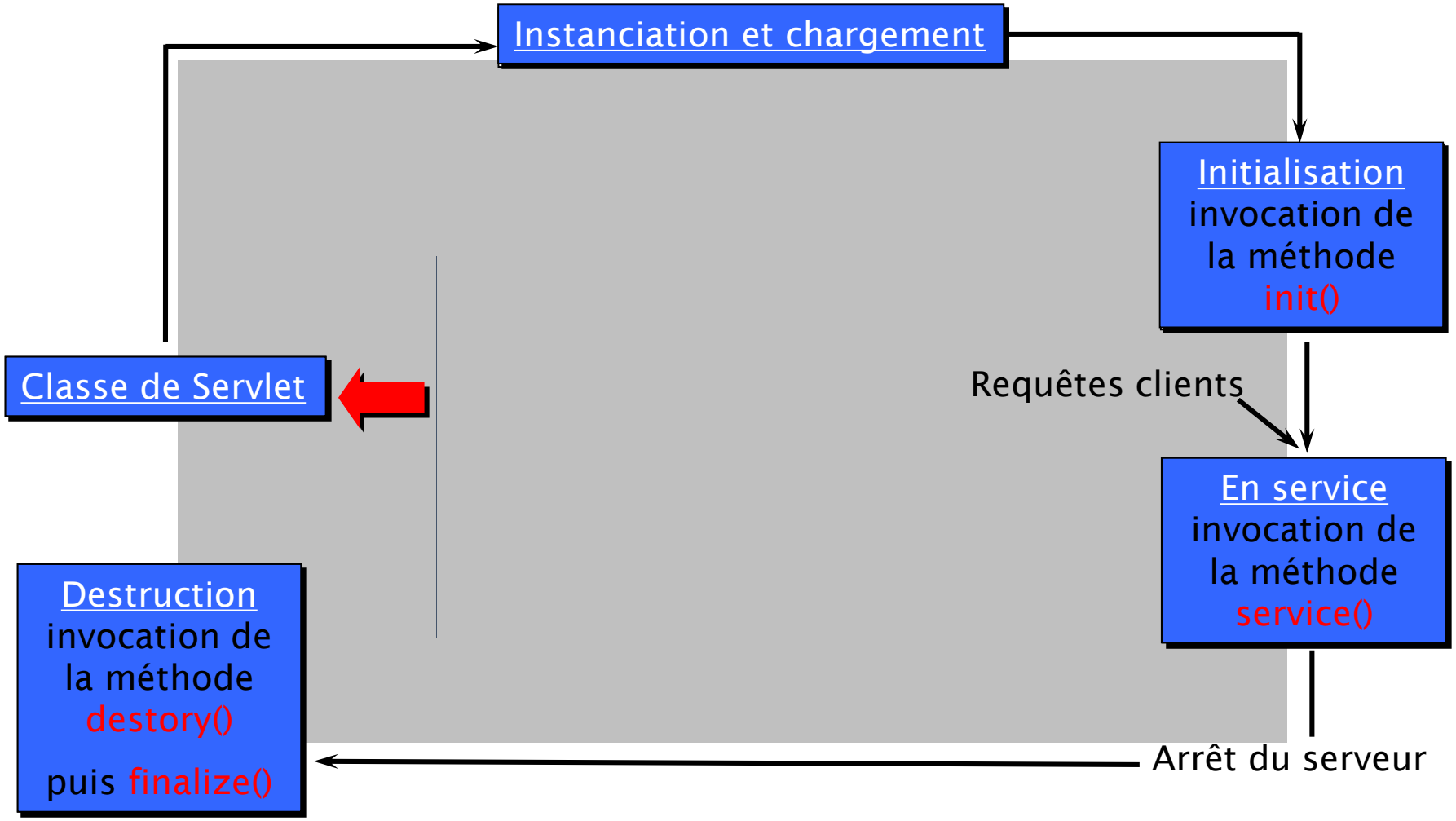
✓ La méthode service() de HttpServlet n'est pas surcharger mais elle prend en charge l'appel automatique de la bonne méthode doXXX() en fonction du type de requêtes.

Modèle de programmation d'une Servlet HTTP (2/2)

Modèle de programmation requête-service-réponse pour une Servlet Http



Cycle de vie d'une Servlet HTTP



Ecrire une Servlet HTTP (1/2)

- La *servlet* doit dériver de la classe **javax.servlet.http.HttpServlet**

- Les méthodes suivantes sont définies:
  - **init()**, **destroy()** et **getServletInfo()**

- Il faut redéfinir une de ces méthodes :
  - **doPost()** ou **doGet()** (ou **doPut()** ou **doDelete() doHead()** ou **doOptions()** ou **doTrace()**) suivant le type de requêtes Http du client que la *servlet* doit traiter : POST, GET, PUT ou DELETE

Ecrire une Servlet HTTP (2/2)

- Utiliser les objets **HttpServletRequest** et **HttpServletResponse** passés en paramètres des méthodes **doGet()** ou **doPost()** pour implémenter le service:

  ✓ **HttpServletRequest** contient les renseignements sur le formulaire HTML initial (utile pour **doPost()**)

  ✓ **HttpServletResponse** contient le flux de sortie pour la génération de la page HTML résultat

  - ce flux de sortie est obtenu par les méthodes :
    - **getWriter()** : recommandé pour retourner du texte
    - **getOutputStream()** : recommandé pour des données binaires

Squelette d'une Servlet HTTP (GET)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet
{
   public void init (HttpServletConfig c)
      throws  ServletException  {…}
   public void doGet (HttpServletRequest req,
                      HttpServletResponse res)
      throws  ServletException, IOException {…}
    public void destroy() {…}
    public String getServletInfo() {…}
}
```

Modèles d'implémentation d'une Servlet HTTP (1/3)

Hello.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet{
public void doGet(HttpServletRequest req,  HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD> <TITLE>Hello</TITLE> </HEAD>" );
    out.println("<BODY>");
    out.println("<H1>Hello Benjamin</H1>" );
    out.println("<BODY> </HTML>");
    }
}
```

Modèles d'implémentation d'une Servlet HTTP (2/3)

- Sans les imports nécessaires le compilateur javac ne peut pas compiler la classe **Hello.java**.
  - L'exception ServletException est définie dans le paquetage javax.servlet.*;
  - L'exception IOException est définie dans le paquetage javax.io;
  - Les objets HttpServletRequest, HttpServletResponse utilisés comme paramètres des méthodes doGet() ou doPost() sont définies dans le paquetage javax.servlet.http.*;
- ServletException, IOException sont des déclarations d'exceptions levées mais non traitées.

Modèles d'implémentation d'une Servlet HTTP (3/3)

- L'instruction **res.setContentType("text/html")** initialise l'objet **res** qui est de type HttpServletResponse comme étant une réponse de type « texte/html », le type MIME standard pour le contenu de pages HTML.
  - ✓ Exemples de types MIME: image/gif, image/jpeg, text/html, text/plain, text/*, */*.

- L'objet **PrintWriter** permet à une Servlet Http de construire la page HTML destinée à l'utilisateur.
  - L'instruction « printWriter out = res.getWriter() » permet de retrouver un flux de sortie « out » (un printWriter) à travers l'objet res pour envoyer le message « Hello Benjamin » au client.

Déclaration d'une Servlet au sein d'une application Web (1/5)

Web.xml

```
<web-app>
  <servlet>
    <servlet-name>……..</servlet-name>
    <servlet-class>……..</servlet-class>
    <init-param>
       < param-name> …….. </param-name>
       < param-value> …….. </param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>…….</servlet-name>
    <url-pattern>/servlet/…….</url-pattern>
  </servlet-mapping>

</web-app>
```

La déclaration d'une Servlet dans une application Web se fait dans le descripteur de déploiement web.xml

Déclaration d'une Servlet au sein d'une application Web (2/5)

- **<web-app> ……. </web-app>:**
  - encapsule l'ensemble des éléments servant à la configuration de l'application Web.

- **<servlet> …… </servlet>:**
  - encapsule l'ensemble des éléments servant à la configuration de chaque Servlet.

- **<servlet-name> …… </servlet-name>:**
  - contient une chaîne de caractère identifiant la Servlet au sein de l'application web.

Déclaration d'une Servlet au sein d'une application Web (3/5)

- **<servlet-class> …… </servlet-class>:**
  - contient le nom complet de la classe de Servlet (package compris).

- **<init-param> …. </init-param> :**
  - Encapsule les paramètres d'initialisation de la Servlet.
  - Chaque éléments <init-param> ….. </init-param> corresspond à un paramètre représenté par une paire nom/valeur avec les éléments :
    - **<param-name> ….. </param-name>**,
    - **<param-value> ….. </param-value>**.

Déclaration d'une Servlet au sein d'une application Web (4/5)

- **<servlet-mapping> …… </servlet-mapping>:**
  - contient des informations permettant de définir la relation entre les URL et les servlets.

- **<servlet-name> …… </servlet-name>:**
  - contient une chaîne de caractère identifiant la Servlet au sein de l'application web.

- **<url-pattern>/servlet/…….</url-pattern>:**
  - définit comment une Servlet est invoquée.

Déclaration d'une Servlet au sein d'une application Web (5/5)

web.xml

```
<web-app>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>Hello</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/servlet/Hello</url-pattern>
  </servlet-mapping>

</web-app>
```

Déclaration de la Servlet « Hello » dans le descripteur de déploiement web.xml

Invocation d'une Servlet à partir d'un navigateur Web (1/7)

- **Invoquer une Servlet c'est utiliser tout d'abord un conteneur Web (exp : Apache TOMCAT) pour sa mise en œuvre.**

- **Déployer la Servlet au sein d'une application Web.**
  - ✓ Crée un nouveau dossier nommée MaWebApp dans le dossier webapps de TOMCAT.
  - ✓ Le dossier MaWebApp doit contenir un sous dossier nommé Web-INF, qui lui-même contient le descripteur de déploiement web.xml et un sous-dossier nommé classes, qui contient le fichier compilé Hello.class de la Servlet.

- **Deux possibilités d'invocation d'une Servlet :**
  - ✓ invocation de la méthode doGet(…),
  - ✓ invocation de la méthode doPost(..).

Invocation d'une Servlet à partir d'un navigateur Web (2/7)

- **1ère Invocation de la méthode doGet(…) :**
  - Saisie de l'URL de la Servlet dans la barre d'adresse du navigateur.
  - http://<hôte>:<port>/<webApp>/servlet/<servlet>
  - http://localhost:8080/MaWebApp/servlet/Hello

Invocation d'une Servlet à partir d'un navigateur Web (3/7)

- **2ème Invocation de la méthode doGet(…):**
  - Clique sur un lien hypertexte qui pointe sur l'URL de la Servlet.

Index.html

```
<HTML>
<HEAD>
  <TITLE>
   Test de la servlet Heloo par clic sur lien
  </TITLE>
 </HEAD>
 <BODY>
  <P>
  <A href="/MaWebApp/servlet/Hello">
   Cliquer pour tester la Servlet Hello
   </A>
    </P>
</BODY>
</HTML>
```

Invocation d'une Servlet à partir d'un navigateur Web (4/7)

❑**Invocation de la méthode doPost(…) :**

- La méthode doPost() d'une Servlet est invoquée principalement lors de l'envoie des données saisies dans un formulaire HTML (par un clic sur un bouton de type submit).

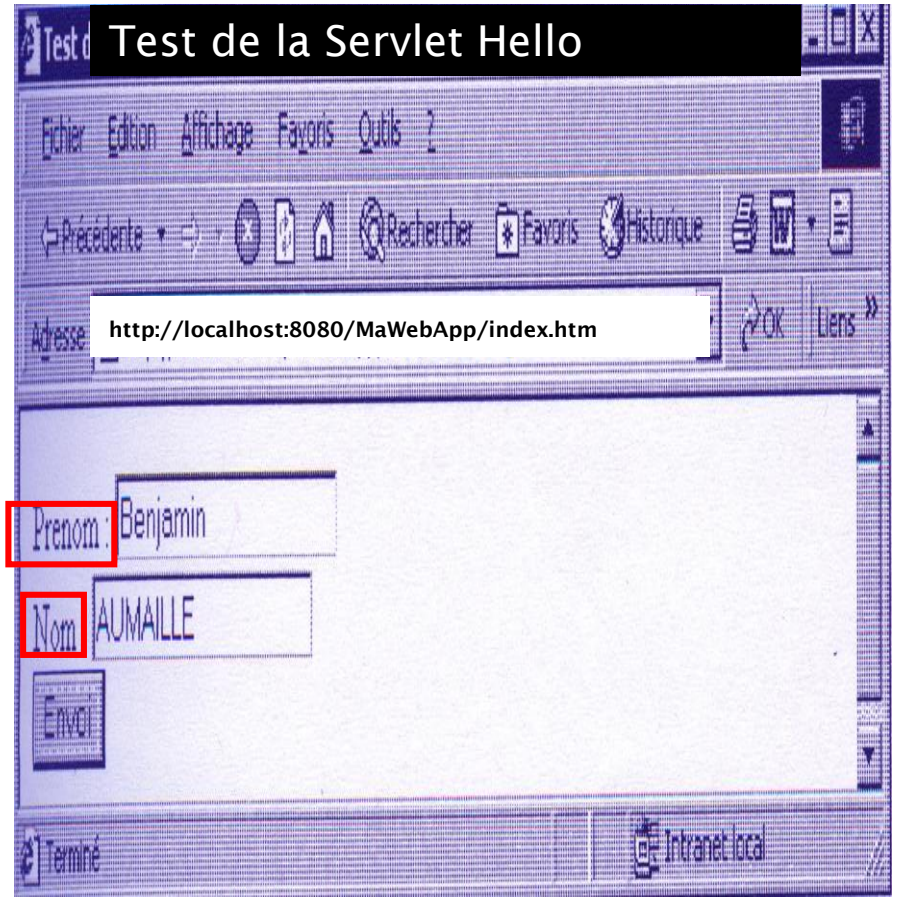
- Exemple de méthode doPost() qui retourne une chaîne de caractères concaténée avec les valeurs des paramètres transmis par le client.

Invocation d'une Servlet à partir d'un navigateur Web (5/7)

❑ **Invocation de la méthode doPost(…) :**

Index.html

```
<HTML>
<HEAD>
   <TITLE> Test de la servlet Hello </TITLE>
</HEAD>

<BODY>
<FORM action = "/test/servlet/Hello" method ="post">
  <P>
  Prenom : <INPUT type ="text" name="prenom">
  <BR>

  Nom : <INPUT type ="text" name="nom">
  <BR>

<INPUT type ="submit" value="Valider">
  </P>

</FORM>
</BODY>
</HTML>
```



Invocation d'une Servlet à partir d'un navigateur Web (6/7)

❑ **Invocation de la méthode doPost(…) :**

Hello.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet{
public void doPost(HttpServletRequest req,  HttpServletResponse
res)
throws ServletException, IOException {

String prenom = req.getParametre("prenom");
String nom = req.getParametre("nom");

res.setContentType("text/html");
PrintWriter out = res.getWriter();
 out.println("<HTML><BODY>");
 out.println("<H1>Bonjour" +prenom+" "+ nom+"."+ "</H1>" );
 out.println("</HTML><BODY> ");
  }
}
```

Invocation d'une Servlet à partir d'un navigateur Web (7/7)

❑**Invocation de la méthode doPost(…) :**

Configuration de Servlet: Interface ServletConfig

- Les informations de configuration d'une Servlet au sein d'une application Web (nom de la Servlet, les paramètres sous formes de nom/valeur) sont représentées par un objet de type javax.servlet.ServletConfig.

- L'objet javax.servlet.ServletConfig est crée par le conteneur web pour chaque élément <servlet> déclaré dans le descripteur de déploiement web.xml de l'application web.

- Les informations de configurations représentées par l'objet javax.servlet.ServletConfig peuvent ensuite être récupérées par la Servlet de préférence lors de sa phase d'initialisation au sein de la redéfinition de la méthode init(…).

Méthodes de l'interface ServletConfig

- **Méthodes de l'interface javax.servlet.ServletConfig dédiées à la récupération des paramètres d'une Servlet:**
  - public **String getServletName()** :
    - Récupérer le nom de la Servlet déclaré au sein du descripteur de déploiement ou le nom de la classe de la Servlet.
  - public **String getInitParameter(String nom)** :
    - Récupérer une chaîne de caractères contenant la valeur d'un paramètre nommé nom ou la valeur null si le paramètre n'existe pas.
  - public **java.util.Enumeration getInitParameterNames()**:
    - Récupérer sous la forme d'un objet de type java.util.Enumeration l'ensemble des noms des paramètres déclarés pour la Servlet.
  - public **ServletContext getServletContext():**
    - Récupérer une référence sur le contexte d'exécution de la Servlet qui permet d' intéragir avec le conteneur web de l'application Web.

Initialisation d'une Servlet: redéfinition de la méthode init() (1/3)

- Syntaxe de la méthode init() :
  - public void init(ServletConfig config) throws ServletExeption;
  - public void init() throws ServletExeption;

web.xml

```
<web-app>
  <servlet>
    <servlet-name>InitServlet</servlet-name>
    <servlet-class>InitServlet</servlet-class>
    <init-param>
       < param-name> param1 </param-name>
       < param-value> value1 </param-value>
    </init-param>

    <init-param>
       < param-name> param2 </param-name>
       < param-value> value2 </param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>InitServlet</servlet-name>
    <url-pattern>/servlet/InitServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Initialisation d'une Servlet: redéfinition de la méthode init() (2/3)

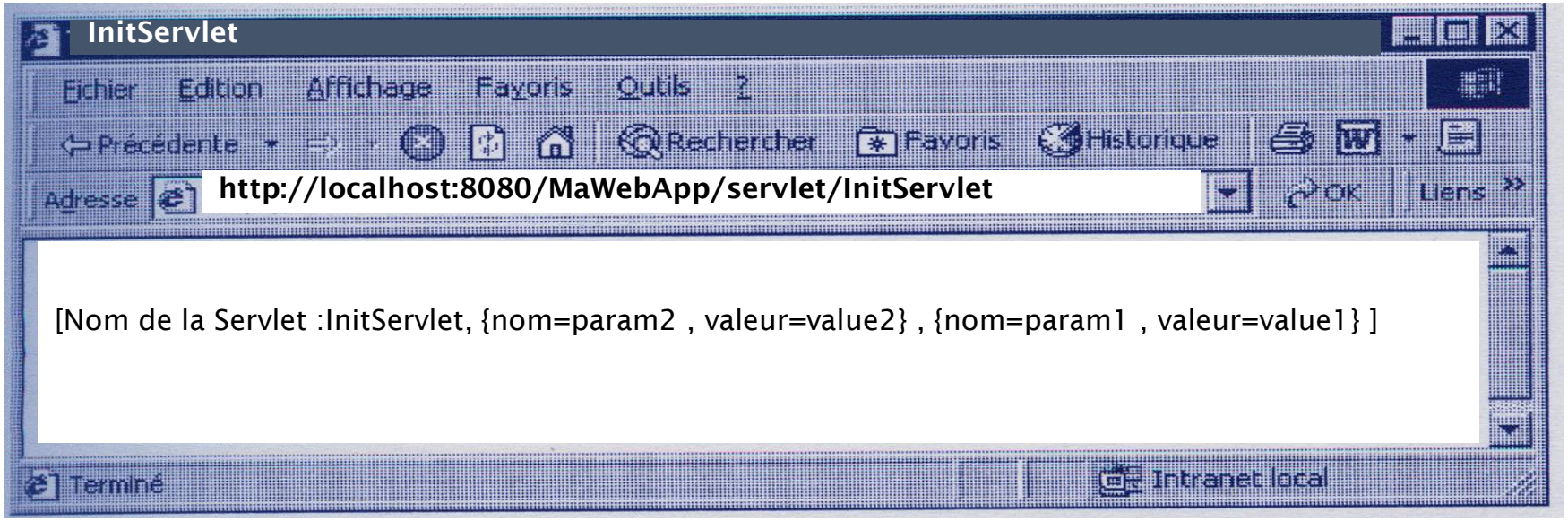
- **Exemple de manipulation:**

InitServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class InitServlet extends HttpServlet{
 Vector vector=new Vector();
 public void init() throws ServletException{
   ServletConfig config =getServletConfig();
    Enumeration lstParams = config.getInitParameterNames( );
    vector.add(" Nom de la Servlet : " + config.getServletName());

    while (lstParams.hasMoreElements() ) {
                String nomParam = (String) lstParams.nextElement();
                vector.add("{nom=" + nomParam+ " , valeur=" +config.getInitParameter(nomParam)+ "} ");
              }
        }
  public void doGet(HttpServletRequest req,  HttpServletResponse res) throws ServletException, IOException {
     PrintWriter out = res.getWriter();
     out.write(vector.toString());
          }
        }
```

Initialisation d'une Servlet: redéfinition de la méthode init() (3/3)

- **Exemple de manipulation (suite):**



Paramètres de l'application Web (1/9)

- De la même manière que pour chaque Servlet d'une application Web, il est possible de déclarer des paramètres globaux (informations de configuration) pour toute l'application Web.

- Ces paramètres peuvent être utiles pour déclarer des informations susceptible d'être utilisées par plusieurs Servlets de l'application Web:
  - ✓ Nom et e-mail de l'administrateur, qui peuvent être utilisés pour générer une page d'erreur à un client.
  - ✓ Nom d'hôte ou adresse IP de machines distantes, qui peuvent être utiles pour l'accès à des ressources distantes.
  - ✓ Nom de la base de données, nom du pilote JDBC à utiliser, nom d'utilisateur et mot de passe pour établir la connexion.
  - ✓ Etc.

Paramètres de l'application Web (2/9)

- Les informations de configuration d'une application Web sont représentées par un objet de type javax.servlet.ServletContext.

  ✓ Chaque Servlet d'une même application Web a donc accès à ces informations.

- L'objet javax.servlet.ServletContext propose des méthodes permettant de travailler principalement avec deux catégories de données :

  ✓ Accéder à des paramètres globaux de l'application Web déclarés dans son descripteur de déploiement web.xml.

  ✓ Créer, lire et supprimer des attributs de façon logicielle, permettant le partage de ressources entre les Servlets d'une même application Web.

Paramètres de l'application Web (3/9)

- **Configuration des paramètres globaux dans le descripteur de déploiement : web.xml**

web.xml

```
<web-app>

<context-param>
     <param-name>……</param-name>
     <param-value>……..</param-value>
</context-param>

<servlet>
     <servlet-name>InitServlet</servlet-name>
     <servlet-class>InitServlet</servlet-class>

     ……..
</servlet>

<servlet-mapping>
     <servlet-name>InitServlet</servlet-name>
     <url-pattern>/servlet/InitServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Paramètres de l'application Web (4/9)

- **Méthodes de l'interface javax.servlet.ServletContext dédiées à la récupération des paramètres globaux d'initialisation:**
  - public String getInitParameter(String nom):
    - Récupérer une chaîne de caractères contenant la valeur d'un paramètre nommé nom ou la valeur null si le paramètre n'existe pas.
  - public java.util.Enumeration getInitParameterNames():
    - Récupérer sous la forme d'un objet de type java.util.Enumeration l'ensemble des noms des paramètres déclarés pour la Servlet.

Paramètres de l'application Web (5/9)

- **Exemple de manipulation:**

web.xml

```
<web-app>
  <context-param>
     <param-name>nomAdmin</param-name>
     <param-value> Yosra ZGUIRA</param-value>
  </context-param>

   <context-param>
     <param-name>emailAdmin</param-name>
     <param-value>yosra.zguira@insa-lyon.fr</param-value>
   </context-param>
.....

<servlet>
     <servlet-name>ErreurServlet</servlet-name>
     <servlet-class>ErreurServlet</servlet-class>
</servlet>
  <servlet-mapping>
     <servlet-name>ErreurServlet</servlet-name>
     <url-pattern>/servlet/ErreurServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Paramètres de l'application Web (6/9)

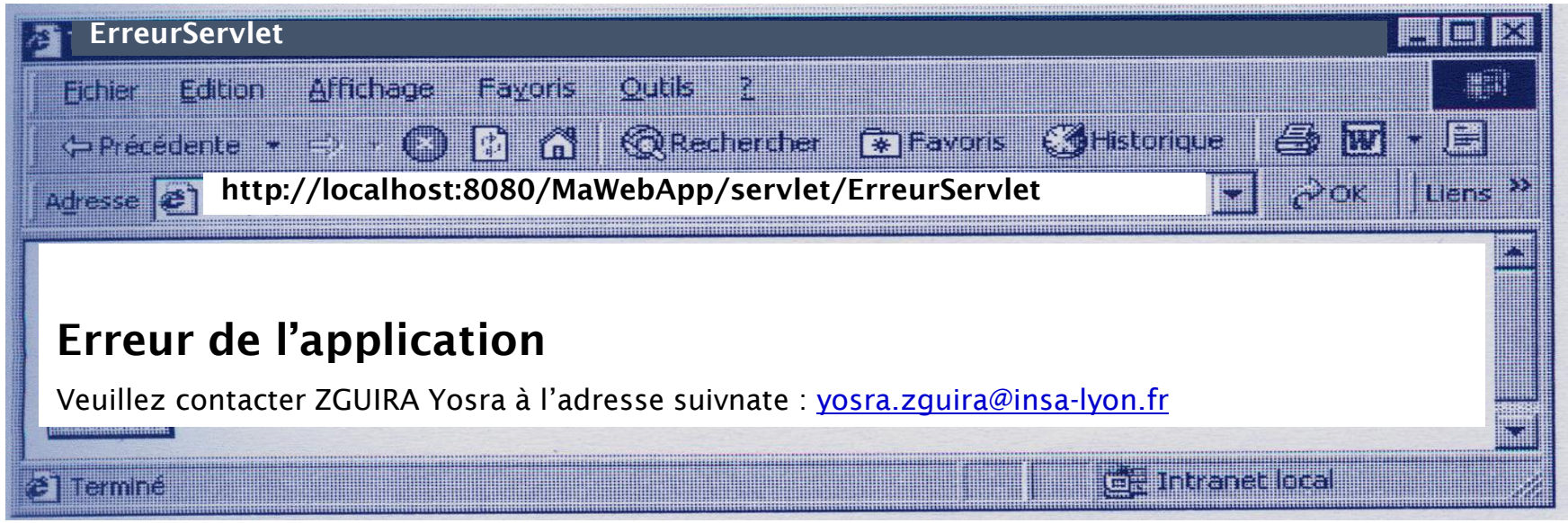
- **Exemple de manipulation (suite):**

ErreurServlet.java

```
import java.io.*;
        import javax.servlet.*;
        import javax.servlet.http.*;

        public class ErreurServlet extends HttpServlet{

        public void doGet(HttpServletRequest req,  HttpServletResponse res) throws ServletException,
        IOException {

        ServletContext application=getServletContext();
        String nom= application.getInitParameter ("nomAdmin");
        String email= application.getInitParameter (" emailAdmin");
        res.setContentType(" text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML><BODY>");
         out.println("<H1>Erreur de l'application</H1>" );
         out.println("<BR><H4>Veuillez contacter <B>" + nom + "</B>" );
         out.println("<A href =\"mailto:"+ email + "\" >"+email+"</A>" );
         out.println("<H4></BODY></HTML> ");
        }
        }
```

Paramètres de l'application Web (7/9)

- **Exemple de manipulation (suite):**



Paramètres de l'application Web (8/9)

- **Méthodes de l'interface javax.servlet.ServletContext dédiées à la gestion logicielle des attributs du contexte d'application:**
  - ✓ public String setAttribute(String nom, Object objet):
    - Créer un attribut dans le contexte de l'application Web. Si le nom de l'attribut existe déjà, la valeur existante est remplacée par la nouvelle.
  - ✓ public Object getAttribute(String nom):
    - Récupérer la valeur d'un attribut dont le nom est passé en paramètre, ou la valeur **null** si l'attribut n'existe pas.
  - ✓ public java.util.Enumeration getAttributeNames():
    - Récupérer sous la forme d'un objet de type java.util.Enumeration le nom de tous les attributs stockées dans l'application Web.
  - ✓ public void removeAttribute(String nom):
    - Supprimer un attribut du contexte de l'application Web, dont le nom est passé en paramètre.

Paramètres de l'application Web (9/9)

- **Exemple de manipulation:**

```
…….

        Employé emp1 = new Employé (" Yosra ", ZGUIRA ");
        Employé emp2 = new Employé ("toto ", "titi ");
        Employé emp3 = new Employé (" tata ", "tatou ");
        ……
        javax.servlet.ServletContext contextApp = getServletContext();

        contextApp.setAttribute(" Employé1 ", emp1);
        contextApp.setAttribute(" Employé2 ", emp2);
        contextApp.setAttribute(" Employé3 ", emp3);
        ……
```

```
…

        javax.servlet.ServletContext contextApp = getServletContext();
        Java.util.Enumeration nomAttributs = contextApp.getAttributeNames( );

        while ( nomAttributs.hasMoreElements() ) {
            String nom = (String) nomAttributs.nextElement();
            Employé e = (Employé) contextApp.getAttribute(nom);
            ……

            contextApp.removeAttribute(nom);
        ……
```

Interfaces ServletRequest et HttpServletRequest (1/7)

- **<u>Méthodes de Récupération d'informations sur l'URL de la requête:</u>**
  - ✓public String getScheme( ):
    - Retourne le nom du protocole utilisé par le client pour émettre sa requête. Par exemple : http, ftp, etc.
  - ✓public String getContextPath( ):
    - Retourne sous la forme d'une chaîne de caractères commençant par un **/,** la portion de l'URL de la requête correspondant au nom du contexte de l'application Web . Par exemple : **/MaWebApp.**
  - ✓public String getMethod():
    - Retourne le nom de la méthode HTTP(GET, POST, etc) utilisée par le client pour émettre sa requête.

Interfaces ServletRequest et HttpServletRequest (2/7)

- **<u>Méthodes de récupération d'informations sur l'URL de la requête (suite):</u>**

  ✓ public String getRequestURL( ):
  - Retourne l'URL que le client a utilisée pour émettre sa requête. L'URL retournée contient le nom du protocole, le nom du serveur, le numéro de port et le chemin d'invocation de la ressource web, mais pas les paramètres de la chaîne de requête. Par exemple: **http://localhost:8080/test/servlet/Hello.**

  ✓ public String getServletPath( ):
  - Retourne la partie de l'URL qui invoque la Servlet/JSP, composée du chemin et du nom ou de l'alias de la Servlet/JSP. Par exemple : **/servlet/Hello.**

Interfaces ServletRequest et HttpServletRequest (3/7)

- **<u>Méthodes de récupération d'informations sur le client:</u>**

  ✓public String getRemoteAddr( ) :
  - Retourne l'adresse IP du client qui a émis la requête. Par exemple **: 127.0.0.1**

  ✓public String getRemoteHost( ):
  - Retourne le nom complet du client qui a émis la requête**.** Par exemple **: 127.0.0.1**

  ✓public String getRemoteUser( ):
  - Retourne le nom de l'utilisateur qui a envoyé la requête si celui s'est authentifié au préalable, sinon retourne la valeur **null**.

Interfaces ServletRequest et HttpServletRequest (4/7)

- **<u>Méthodes de récupération d'informations sur le serveur:</u>**

  ✓public String getServerName( ):
  - Retourne le nom d'hôte du serveur qui a reçu la requête. Par exemple **: localhost**

  ✓public String getServerPort( ):
  - Retourne le numéro de port d'écoute du serveur qui a reçu la requête**.** Par exemple **: 8080**

Interfaces ServletRequest et HttpServletRequest (5/7)

- **Méthodes de récupération d'informations dans l'en-tête HTTP:**

  ✓public String getHeader(String nom):
  - Retourne la valeur de l'entête nommé, passé en paramètre ou la valeur **null** si l'entête n'existe pas. Le nom de l'entête est sensible à la casse. Par exemple **: getHeader("Accept-Language")** retourne **fr.**

  ✓public java.util.Enumeration getHeaders(String nom):
  - Retourne sous la forme d'un objet de type java.util.Enumeration l'ensemble des valeurs de l'en-tête de la requête spécifié en paramètre.

  ✓public java.util.Enumeration getHeaderNames():
  - Retourne sous la forme d'un objet de type java.util.Enumeration l'ensemble des noms des en-têtes contenus dans la requête.

Interfaces ServletRequest et HttpServletRequest (6/7)

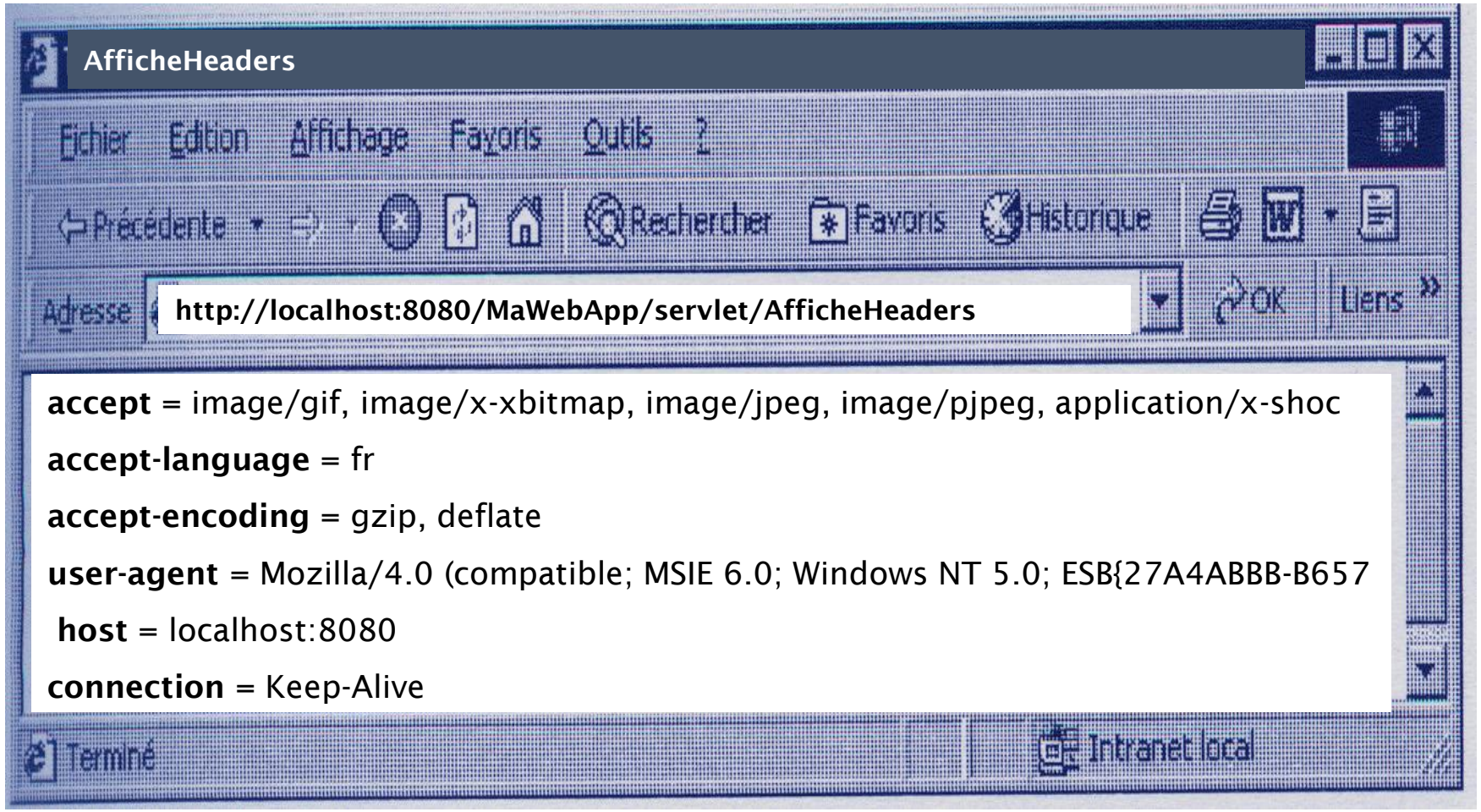
- **Exemple de manipulation:**

AfficheHeaders.java

```
 import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;
 import java.util.*;

public class AfficheHeaders extends HttpServlet{

public void doGet(HttpServletRequest req,  HttpServletResponse res) throws ServletException, IOException {
 PrintWriter out = res.getWriter();
 res.setContentType("Text/plain");
Enumeration enum = req.getHeaderNames( );

 while (enum.hasMoreElements() ) {
                String headerNom = (String) enum.nextElement();
                out.println(headerNom+" = "+ req.getHeader(headerNom));
        }

  }
}
```

Interfaces ServletRequest et HttpServletRequest (7/7)

- **Exemple de manipulation (suite):**



Interfaces ServletResponse et HttpServletResponse (1/7)

- **Méthodes de déclaration du type du contenu et de la taille de la réponse:**

✓ public void setContentType(String type):

- Spécifier le type MIME de contenu du corps de la réponse HTTP**.** Par exemple text/html pour du HTML, text/plain pour du texte brut, application/pdf pour un document Adobe pdf …

✓ public void setContentLength(int taille):

- Spécifier la taille du contenu de la réponse HTTP. Autrement dit définir l'en-tête HTTP Content-Length.

Interfaces ServletResponse et HttpServletResponse (2/7)

- **Méthodes de renseignement des informations dans l'en-tête HTTP:**

✓ public void setHeader(String nom, String Valeur):

- Initialiser un en-tête dans la réponse HTTP, avec le nom et la valeur spécifiés en paramètres. Si l'en-tête existe déjà, la nouvelle valeur remplace l'ancienne.

✓ public void addHeader(String nom, String Valeur):

- Ajouter un en-tête dans la réponse HTTP, avec le nom et la valeur spécifiés en paramètres. Cette méthode permet à un en-tête d'avoir plusieurs valeurs.

✓ public boolean containsHeader(String nom):

- Retourne un booléen indiquant si un entête existe ou non.

Interfaces ServletResponse et HttpServletResponse (3/7)

- **Méthodes d'envoi d'erreurs et d'états HTTP:**

✓ public void sendError(int sc) throws java.io.IOException
  - Envoyer une réponse d'erreur au client en utilisant le code d'état (status code) correspondant et effacer la mémoire tampon (buffer).

✓ public void sendError(int sc,String message) throws java.io.IOException
  - Envoyer un code d'erreur HTTP au client. Par exemple **SC-NOT-FOUND(404)** ou **SC-SERVICE-UNAVAILABLE(503)**.

✓ public void sendStatus(int sc):
  - Appliquer un code d'état à la réponse HTTP quand il n'y a pas d'erreur, comme par exemple **SC-OK(200)** ou **SC-CONTINUE(100)**.

Interfaces ServletResponse et HttpServletResponse (4/7)

- **Méthodes de redirection d'URL:**

  ✓public void sendRedirect(String url) throws java.io.IOException

  - Envoyer au navigateur du client un ordre de redirection sur une autre ressource Web (servlet, jsp, fichier html), qui peut être de la même application Web ou non.

  - L'URL de la ressource Web passée en paramètre peut être relative ou absolue.

  - Exemple d'URL relative :
    - res.sendRedirect("/MaWebApp/indentification.html")

  - Exemple d'URL absolue :
    - res.sendRedirect("http://www.lyon.com");

Interfaces ServletResponse et HttpServletResponse (5/7)

- **<u>Méthodes pull client:</u>**

  ✓ Le pull client est similaire à la redirection, avec une différence principale : le navigateur affiche le contenu de la première page et attends un certain temps avant de retrouver et afficher le contenu de la page suivante.

  ✓ **Utilités:**

  - Le contenu de la première page peut expliquer au client que la page demandée a été déplacée avant que la page suivante ne soit automatiquement chargée.

  - Les pages peuvent être retrouvées en séquence, rendant ainsi possible une animation de mouvements lente.

Interfaces ServletResponse et HttpServletResponse (6/7)

- **Méthodes pull client (suite):**
  - ✓ L'information de pull client est envoyée au client via l'en-tête HTTP Refresh.
  - ✓ La valeur de cet en-tête indique le nombre de secondes pendant lesquelles la page doit être affichée avant d'aller chercher la prochaine et elle peut aussi inclure l'URL indiquant où aller la chercher.
  - ✓ res.SetHeader("Refresh", "3");
    - Indique au client de recharger la même Servlet après avoir affiché son contenu courant pendant trois secondes.
  - ✓ res.SetHeader("Refresh", "3;URL=http://www.lyon.com");
    - Indique au client d'afficher la page d'accueil Lyon après trois secondes.

Interfaces ServletResponse et HttpServletResponse (7/7)

- **Exemple de manipulation : Mise à jour de l'heure courante**

ClientPull.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class ClientPull extends HttpServlet{

public void doGet(HttpServletRequest req,  HttpServletResponse res) throws ServletException, IOException {
res.setContentType("Text/plain");
PrintWriter out = res.getWriter();

res.setHeader ("Refresh", "60");

out.println(new Date().toString());

  }
}
```

Synchronisation des traitements: L'Interface SingleThreadModel (1/4)

- **Par défaut une Servlet fonctionne dans un environnement multitâche.**

  ✓C'est-à-dire qu'à chaque requête reçue pour une Servlet, le conteneur Web crée un thread qui va exécuter la méthode de service d'une instance de la Servlet.

  ✓Si la méthode de service travaille avec des variables d'instance de la Servlet; chaque thread peut modifier la valeur de ces variables indépendamment de la logique de traitement des autres threads.

- **Obligation de garantir dans un certain cas un fonctionnement isolé de chaque thread.**

Synchronisation des traitements: L'Interface SingleThreadModel (2/4)

- **Exemple d'application posant un problème :**
  - Deux client désirent s'enregistrer sur votre site par l'intermédiaire d'un formulaire HTML. Ils envoient leurs données en même temps à destination d'une Servlet dont le rôle est de créer un enregistrement dans la tables clients d'une base de données.

  - La Servlet doit donc procéder en deux étapes :
    1. Récupérer la plus grande valeur de clé primaire actuellement présente dans la Table.
    2. Créer un nouvel enregistrement dans la table avec les données du client, en donnant à la clé primaire la valeur maximale récupérée précédemment, plus un.

  - **Que se passe-t-il si deux instances de la Servlet effectuent la première étape du traitement en même temps ?**
    - Une seule des deux instances pourra exécuter la deuxième étape. L'autre instance obtient une erreur de la base de données, car elle tente de créer un doublon de clé primaire. Un seul des deux clients est donc enregistré sur votre site.

Synchronisation des traitements: L'Interface SingleThreadModel (3/4)

- **Solution du problème :**
  - ✓ Avec l'implémentation de l'interface javax.servlet.SingleThreadModel, un conteneur Web prend en charge le fait qu'une instance de Servlet ne peut être exécutée par un plusieurs threads à la fois.

```
import javax.servlet.*;
.....
public class MaServlet extends HttpServlet implements SingleThreadModel
{.....}
```

  - ✓ L'utilisation de l'interface javax.servlet.SingleThreadModel implique que le conteneur Web invoque la méthode de service dans un **bloc synchronisé**.

  - ✓ Bien souvent, seules quelques instructions sont réellement critiques.
    - Au lieu de synchroniser toute la méthode service, il est possible d'améliorer les performances en synchronisant uniquement les quelques instructions sensibles en utilisant tout simplement un ou plusieurs blocs synchronisés.

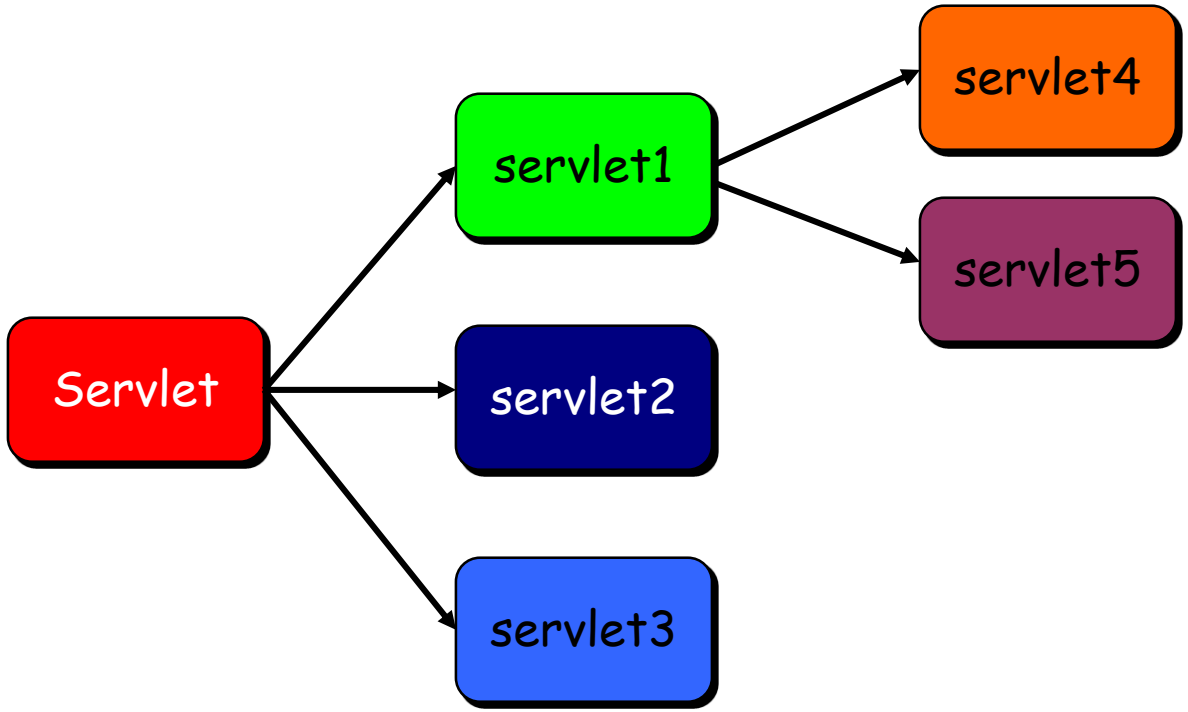
Synchronisation des traitements: L'Interface SingleThreadModel (4/4)

- **Exemple de syntaxe d'un bloc synchronisé dans une Servlet:**

```
 .........
Object obj = new Object();

public void doXXXX(HttpServletRequest req,  HttpServletResponse res)
throws ServletException, IOException {
.........

Synchronized(obj) {
.............
}

  }
}
```

Collaboration entre Servlets: L'Interface RequestDispatcher (1/2)

- **Agrégation de résultats fournis par des Servlets:**
  - ✓ meilleure modularité,
  - ✓ meilleure réutilisation.



Collaboration entre Servlets: L'Interface RequestDispatcher (2/2)

- **Obtention d'un RequestDispatcher :**
  - ✓ dans la méthode de traitement de requête de Servlet

```
……..
RequestDispatcher rd;
rd = getServeletContext().getRequestDispatcher("/servlet/MaServlet");
if(rd==null) res.sendError(404);
        ……..
```

- **Redirection d'une requête:**
  - ✓ dans la méthode de traitement de requête, demande à une autre Servlet de répondre au client

```
rd.forward(req, res);
```